Spring Boot

* Spring Boot is a Spring Project.
* It is used to create a stand alone project. This would be a Spring based application that are quickly configured and can run “out of the box”.
* Spring Boot is highly opinionated: It makes decision about how an application should be set up, so you don’t have to.
  + Spring Boot has a mindset of “convention over configuration”. So if you follow standard conventions, Spring Boot will likely be able to interpret what you are intending to do, so little to no configuration is needed. You can always override configurations.
* Benefits and Features of Spring Boot:
  + Provides “starter dependencies” – These are preconfigured and maintained dependencies that are designed to seamlessly integrate with Spring Boot, adopting the standard configurations so they don’t have to be declared.
  + Configuration, if necessary, is centralized in application.properties, or application.yaml. This XML configuration is needed.
  + **Spring Boot Actuator** – Provides endpoints that give information about our application. This is an actual web site.
  + **Spring Boot DevTools** – Provide additional resources to developers. Many IDEs have similar support like **Live Reload and no-caching.**
  + Embedded Tomcat web server – This is included when you use the Spring MVC starter dependency. This means that each Spring Boot project comes with its own Tomcat server, meaning we no longer have to use the locally installed Tomcat and the application can be built as a JAR instead of a WAR.
* Running a Spring Boot application:
  + To run a spring Boot application, you simply need to the dependencies (usually comes from Spring Initializr) and the @SpringBootApplication annotation above the main method. The main method will also need to call SpringApplication.run
  + @SpringBootApplication - Combines the functionalities @Configuration, @EnableAutoCongiuration, and @ComponentScan.

Spring MVC

* SpringMVC abstracts away Java Servlets. It provides tools for using the MVC design pattern. Its used to create RESTful APIs with Spring.
* SpringMVC is a Spring Module, so it can be used outside of Spring Boot. However, that is not recommended any longer. If you use SpringMVC outside of Spring Boot it will require external web server, and other lifecycle configuration to link the spring Container with the Servlets lifecycle.
* Annotations:
  + **@Controller** 
    - This is the stereotype annotation for our Controllers. These are the methods that will handle the information from the HTTP request and set up the information for the HTTP response.
      * Note: Controllers in SpringMVC are not themselves Servlets. Instead they are classes that receive the necessary information from requests and send the necessary information to set responses, so they don’t always get the full Servlet objects for those things.
  + @RequestMapping
    - Specifies the URL path and HTTP verb that a certain method, or class, in the Controller is layer will handle.
      * There are sub-annotations that are verb specific. Like @GetMapping to label a method that will handle a GET request.
  + @CrossOrigin
    - This will add a CORs filter for requests from the origin specified. Used to bypass CORs errors.
  + @PathVariable
    - Allows you to get a path variable from the request as a parameter to your method.
  + @ResponseBody
    - parses the JSON body of a request as an object passed into the method as a parameter, similar to @PathVariable
  + @RestController
    - this combines @Controller and @ResponseBody annotations.
  + @ResponseStatus
    - Used to indicate that the returned value of the controllers method will be placed within the body of the web response.
  + @ExceptionHandler –
    - Used to send a custom response back to the client when a controller method throws an exception.

SpringMVC Objects

* DispatcherServlet (Servlet object) – This is the FrontController servlet. It is handled by the framework and is not directly seen by the developers.
  + FrontController (design pattern) is where all the requests are filtered through as a single servlet/handler. Any requests that comes in is read for the URL and verb to determine the class that should handle that request.
* HandlerMapping – This interface is used by the DispatcherServlet to define the routes that requests are sent through.
* ViewResolver – Helps build views based on templates, if they are created.

SpringMVC Application Flow:

Diagram

Description automatically generated

1. Tomcat Receives a request from the client.
2. Tomcat passes the request to the FrontController/DispatcherServlet
3. The DispatcherServlet consult the HandlerMapping with the requests URI and HTTP method to determine the controller that should handle the request.
4. DispatcherServlet passes the relevant information to the controller.
5. Controller method runs, interacts with Service Layer, that interacts with data layer.
6. The response data is handed back to the FrontController.
7. The DispatchServlet consults with the ViewResolver if necessary to process views.
8. The DispatcherServlet hands the response to the web container and the HTTP lifecycle continues.